# NetLogger: Distributed System Monitoring and Analysis

Brian L. Tierney

Ernest Orlando Lawrence Berkeley National Laboratory

# Outline

- Overview
  - What is NetLogger?
  - What is NetLogger good for?
  - What is NetLogger not good for?
- NetLogger Components
  - message format
  - instrumentation library
  - system monitoring tools
  - visualization tools
- Case Studies
  - Radiance luminosity application
  - Parallel remote data server (DPSS)
- Current Work
- Current Issues

# Overview

- **The Problem**
  - **When building distributed systems, we often observe unexpectedly low performance**
    - **the reasons for which are usually not obvious**
  - **The bottlenecks can be in any of the following components:**
    - the applications
    - the operating systems
    - the disks or network adapters on either the sending or receiving host
    - the network switches and routers, and so on
- **The Solution:**
    - **Highly instrumented systems with precision timing information and analysis tools**

# Bottleneck Analysis

- Distributed system users and developers often assume the problem is network congestion
  - This is often not true
- In our experience tuning distributed applications, performance problems are due to:
  - network problems: 40%
  - host problems: 20%
  - application design problems/bugs: 40%
    - 50% client , 50% server
- Therefore it is equally important to instrument the applications

# NetLogger Toolkit

- We have developed the NetLogger Toolkit
  - A set of tools which make it easy for distributed applications to log interesting events at every critical point
  - NetLogger also includes tools for host and network monitoring

- The approach is novel in that it combines network, host, and application-level monitoring to provide a complete view of the entire system

# Why “NetLogger”?

- The name “NetLogger” is somewhat misleading
  - Should really be called: “Distributed Application, Host, and Network Logger”

- “NetLogger” was a catchy name that stuck

# When to use NetLogger

- When you want to:
  - do performance/bottleneck analysis on distributed applications
  - determine which hardware components to upgrade to alleviate bottlenecks
  - do real-time or post-mortem analysis of applications
  - correlate application performance with system information (ie: TCP retransmission's)
- works best with applications where you can follow a specific item (data block, message, object) through the system

# When NOT to use NetLogger

- Analyzing massively parallel programs (e.g.: MPI)
  - Current visualization tools don’t scale beyond tracking about 20 types of events at a time

- Analyzing many very short events
  - system will become overwhelmed if too many events
  - we typically use NetLogger to monitor events that take > .5 ms
  - e.g: probably don’t want to use to instrument the UNIX kernel

# NetLogger Components

- **NetLogger Toolkit contains the following components:**
  - **NetLogger message format**
  - **NetLogger client library**
  - **NetLogger visualization tools**
  - **NetLogger host/network monitoring tools**

- **Additional critical component for distributed applications:**
  - **NTP (Network Time Protocol) is required to synchronize the clocks of all systems**

# NetLogger Message Format

- We are using the IETF draft standard Universal Logger Message (ULM) format (http://www.ietf.org/internet-drafts/draft-abela-ulm-05.txt):
  - a list of “field=value” pairs
  - required fields: DATE, HOST, PROG, and LVL
    - LVL is the severity level (Emergency, Alert, Error, Usage, etc.)
  - followed by optional user defined fields
- NetLogger adds these required fields:
    - NL.EVNT, a unique identifier for the event being logged
      - e.g.: SERVER_IN, VMSTAT_USER_TIME, NETSTAT_RETRANSSEG

# NetLogger Message Format

- **Sample NetLogger ULM event:**

```
DATE=19980430133038.55784 HOST=foo.lbl.gov
  PROG=testprog LVL=Usage NL.EVNT=SEND_DATA
  SEND.SZ=49332
```

  - **This says program named *testprog* on host *foo.lbl.gov* performed event named SEND_DATA, size = 49332 bytes, at the date/time given**

- **User-defined data elements (any number) are used to store information about the logged event - for example:**
  - **NL.EVNT=SEND_DATA SEND.SZ=49332**
    - **the number of bytes of data sent**
  - **NL.EVNT=NETSTAT_RETRANSSEGS NS.RTS=2**
    - **the number of TCP retransmits since the previous event**

# Other Formats

- We'd like to convince everyone to use the ULM/NetLogger format for logging
  - This way we can all share log file management and visualization tools
- Probably not realistic
  - Working on filters to convert the following to/from NetLogger format
    - Pablo, NWS. Surveyor?, others?
  - Also working on a binary representation for more efficient use of network and disk
- If ULM is not adequate, whose format is better?

# NetLogger API

- NetLogger Toolkit includes application libraries for generating NetLogger messages
  - Can send log messages to:
    - file
    - host/port (netlogd)
    - syslogd
    - memory, then one of the above
- C, C++, Java, Perl, and Python APIs are currently supported

# NetLogger API

- Only 6 simple calls:
  - NetLoggerOpen()
    - create NetLogger handle, specify logging destination
  - NetLoggerWrite()
    - get timestamp, build NetLogger message, send to destination
  - NetLoggerGTWrite()
    - must pass in results of Unix gettimeofday() call
  - NetLoggerFlush()
    - flush any buffered message to destination
  - NetLoggerSetLevel()
    - set ULM severity level
  - NetLoggerClose()
    - destroy NetLogger handle

# Sample NetLogger Use

```
lp = NetLoggerOpen(method, progname, NULL,
                hostname, NL_PORT);

while (!done)
{
      NetLoggerWrite(lp, "EVENT_START",
                      "TEST.SIZE=%d", size);

      /* perform the task to be monitored */
      done = do_something(data, size);

      NetLoggerWrite(lp, "EVENT_END");
}
NetLoggerClose(lp);
```

# NetLogger Host/Network Tools

- Wrapped UNIX network and OS monitoring tools to log “interesting” events using the same log format
  - *netstat* (TCP retransmissions, etc.)
  - *vmstat* (system load, paging, etc.)
  - *iostat* (disk activity)
  - *ping*

- These tools have been wrapped with Perl or Java programs which:
  - parse the output of the system utility
  - build NetLogger messages containing the results

# NetLogger Network Tools

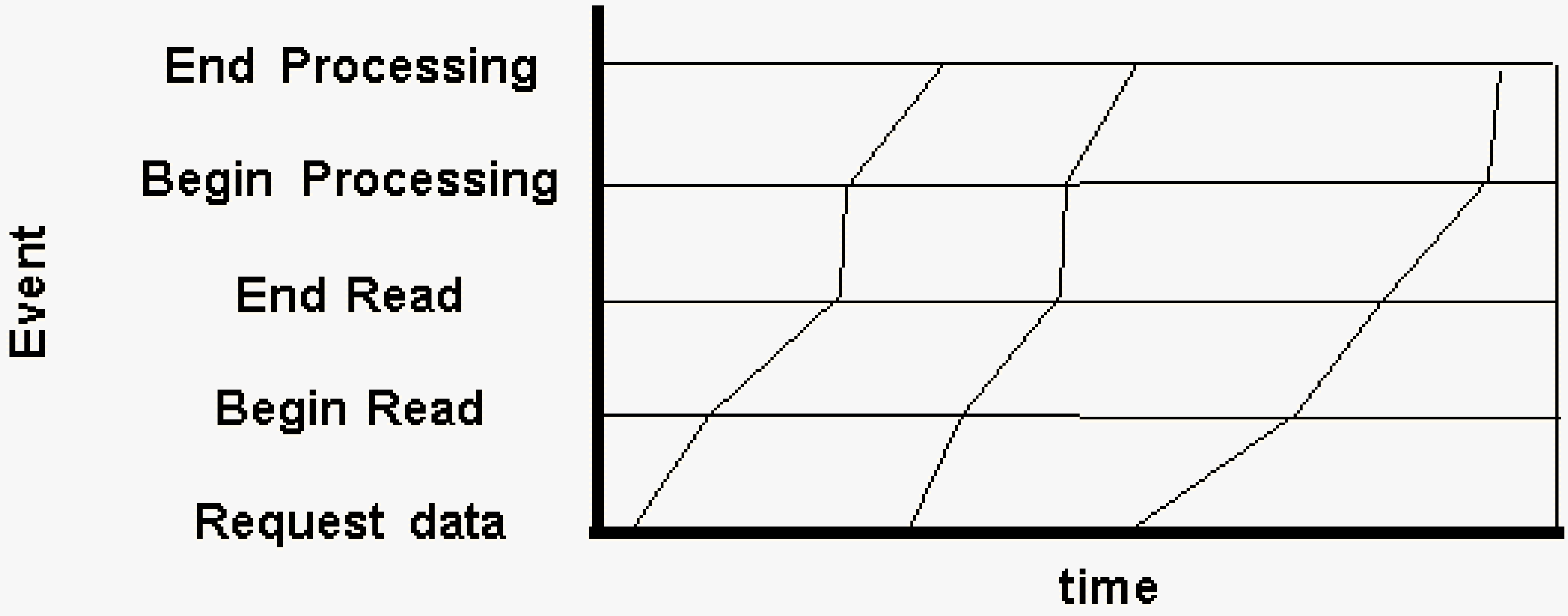
- NetLogger tool for SNMP queries
  - Usage: nl_snmpget hostname object [port]

- Examples:
  - host monitoring
    - `nl_snmpget unix_host sysName`
      - Returns: system.sysName.0 = wakko.lbl.gov
  - router monitoring
    - `nl_snmpget routername ipInDelivers 3`
      - Returns: tcp.tcpInErrs.3 = 4000
  - ATM switch monitoring
    - `nl_snmpget switchname sonetLineFEBEs`
    - `nl_snmpget switchname portTransmittedCells`

# NetLogger Events

- Logged events are correlated with system behavior to characterize the performance of the system during actual operation
  - facilitates bottleneck identification
- Using “life-lines” to visualize the data flow is the key to easy interpretation of the results.
- We believe this type of monitoring is a critical component to building reliable high performance data intensive systems

# NetLogger Event “Life Lines”



# Event Id

- In order to associate a group of events into a “lifeline”, you must assign an event ID to each NetLogger event
- Sample Event Ids
  - file name
  - block ID
  - frame ID
  - user name
  - host name
  - etc.

# Sample NetLogger Use

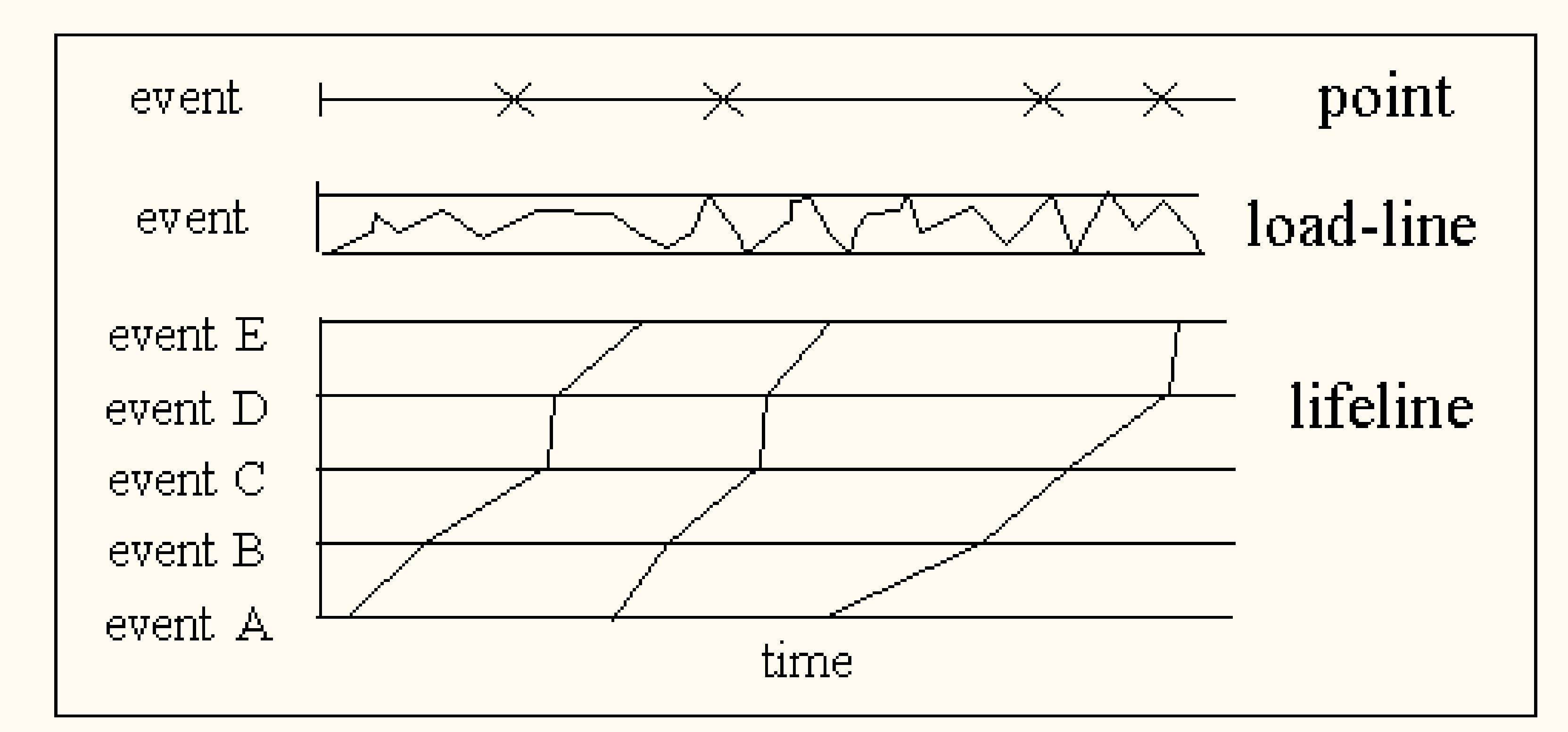
```
lp = NetLoggerOpen(method, progname, NULL, hostname, NL_PORT);
for (i=0; i< num_blocks; i++)  {
    NetLoggerWrite(lp, “START_READ”,
      “BLOCK_ID=%d BLOCK_SIZE=%d”, i, size);
    read_block(i);
    NetLoggerWrite(lp, “END_READ”,
      “BLOCK_ID=%d BLOCK_SIZE=%d”, i, size);
    NetLoggerWrite(lp, “START_PROCESS”,
      “BLOCK_ID=%d BLOCK_SIZE=%d”, i, size);
    process_block(i);
    NetLoggerWrite(lp, “END_PROCESS”,
      “BLOCK_ID=%d BLOCK_SIZE=%d”, i, size);
    NetLoggerWrite(lp, “START_SEND”,
      “BLOCK_ID=%d BLOCK_SIZE=%d”, i, size);
    send_block(i);
    NetLoggerWrite(lp, “END_SEND”,
      “BLOCK_ID=%d BLOCK_SIZE=%d”, i, size);
}
NetLoggerClose(lp);
```

# NetLogger Visualization Tools

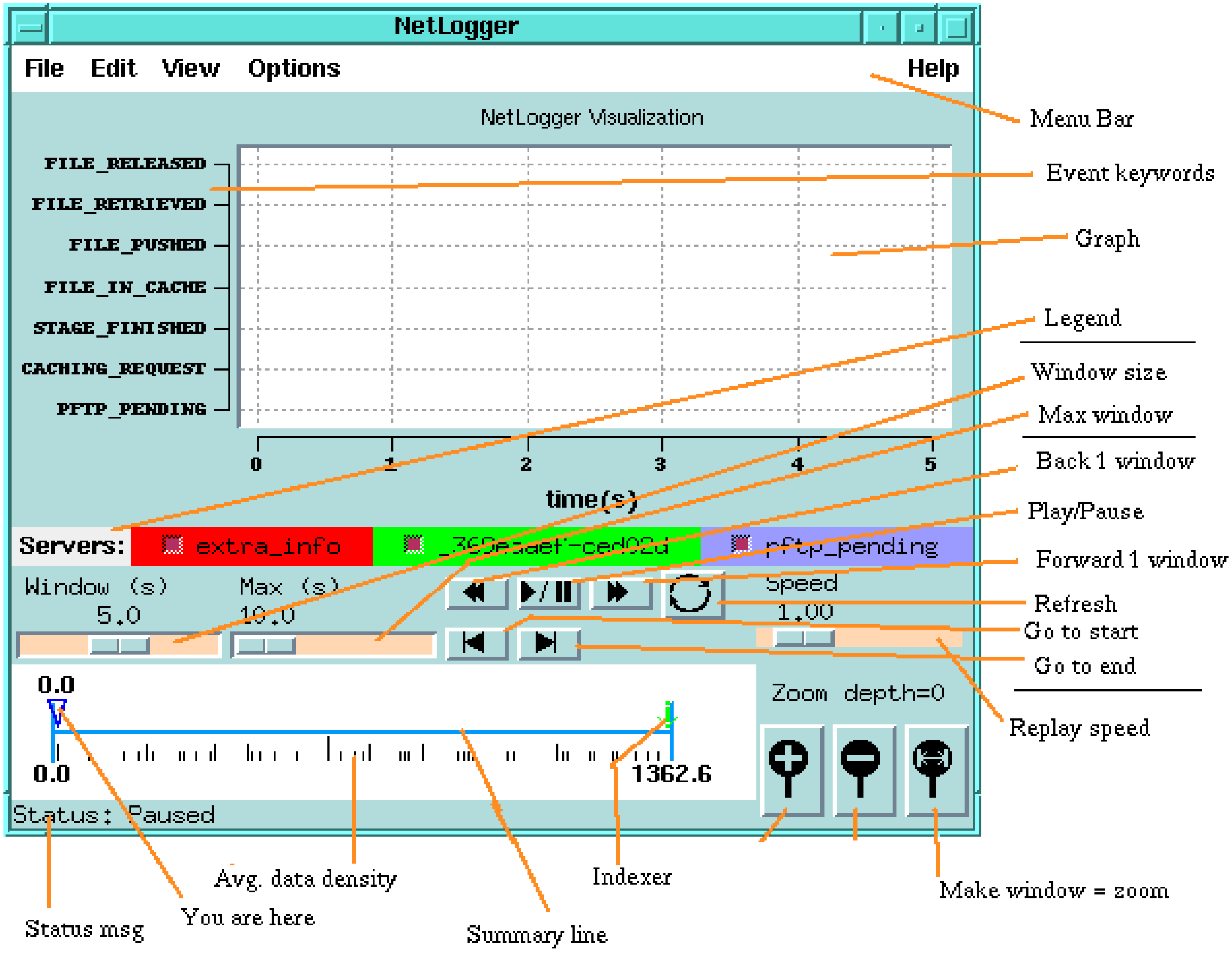
- Exploratory, interactive analysis of the log data has proven to be the most important means of identifying problems
  - this is provided by *nlv* (NetLogger Visualization)
- *nlv* functionality:
  - can display several types of NetLogger events at once
  - user configurable: which events to plot, and the type of plot to draw (lifeline, load-line, or point)
  - play, pause, rewind, slow motion, zoom in/out, and so on
  - *nlv* can be run post-mortem or in real-time
    - real-time mode done by reading the output of *netlogd* as it is being written

# NLV Graph Types

- nlv supports graphing of “points”, “load-lines”, and “lifelines”



# NLV



NetLogger

File Edit View Options Help

NetLogger Visualization

Servers: extra_info _369e5aef-ced02d pftp_pending

Window (s) 5.0

Max (s) 10.0

Speed 1.00

Zoom depth=0

Status: Paused

Menu Bar

Event keywords

Graph

Legend

Window size

Max window

Back 1 window

Play/Pause

Forward 1 window

Refresh

Go to start

Go to end

Replay speed

Make window = zoom

Indexer

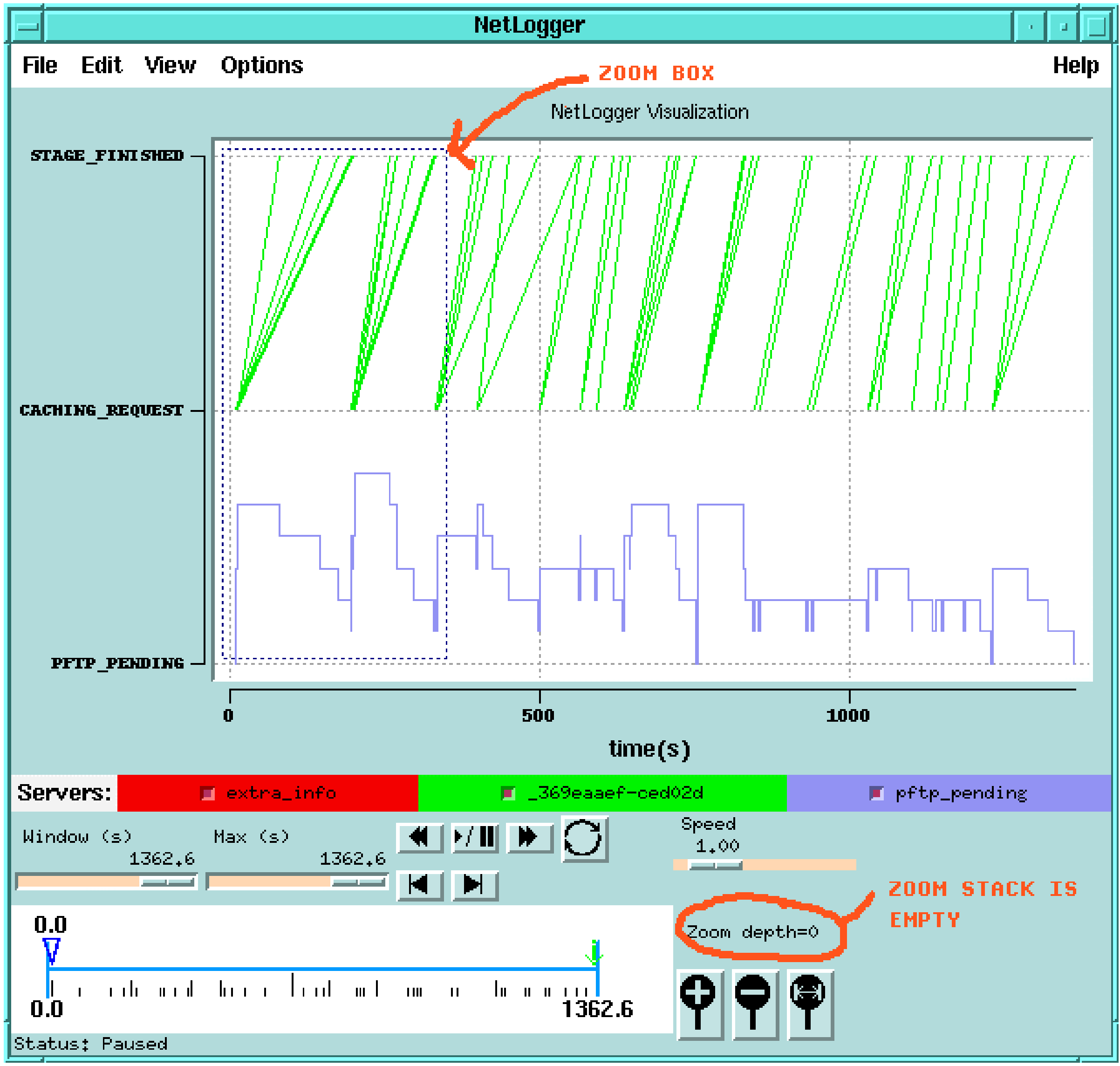
Avg. data density

Summary line

You are here

Status msg

# NLV Zoom Feature



ZOOM BOX

Window (s) 1362.6

Max (s) 1362.6

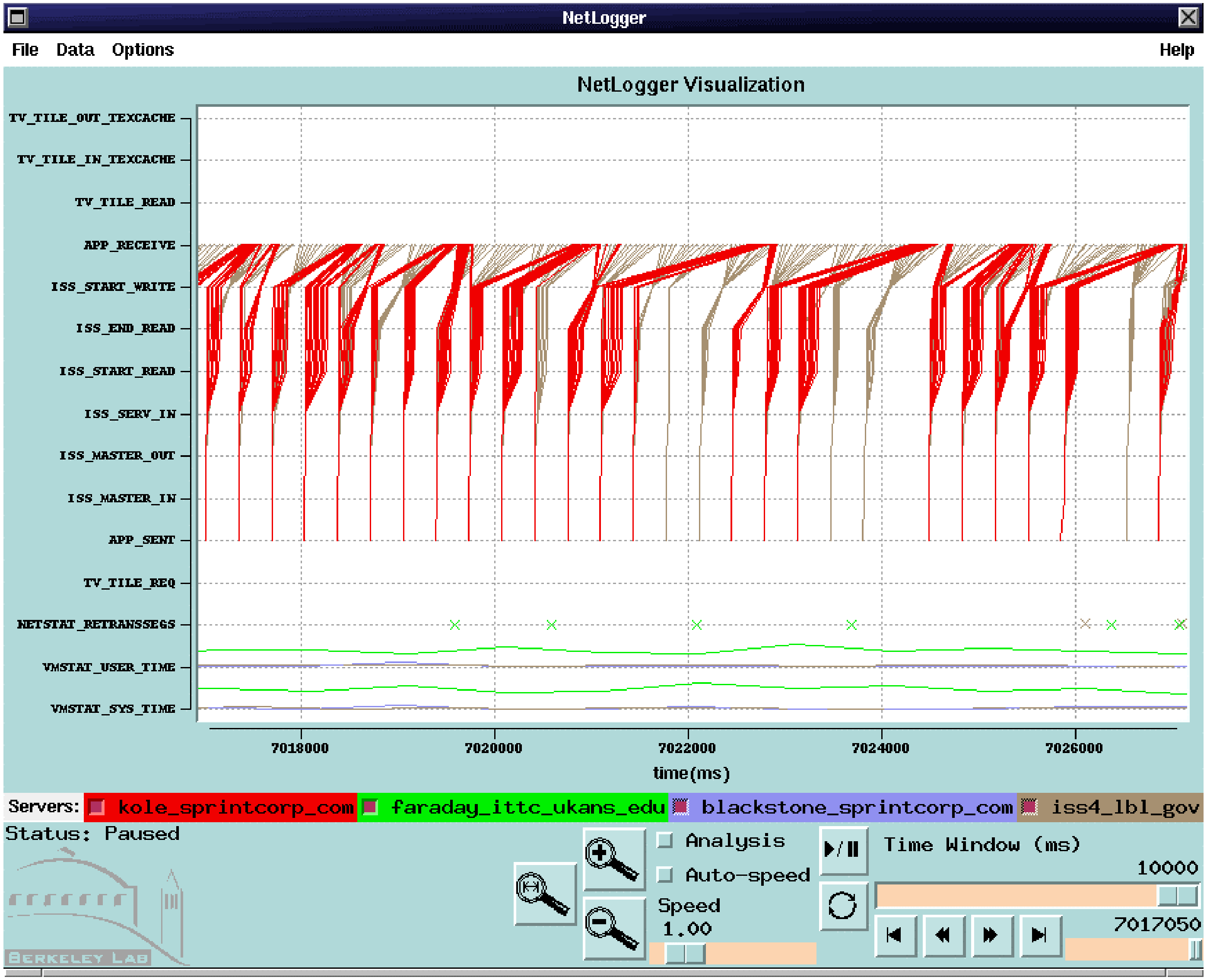
Speed 1.00

Zoom depth=0

ZOOM STACK IS EMPTY

Status: Paused

# NLV with lifeline, load-line, and point events



Status: Paused

□ Analysis

□ Auto-speed

Speed 1.00

Time Window (ms) 10000

7017050

# Example NLV Configuration

```
# display server data as a “lifeline”
set +SERVER_READ
type line

# lifeline constructed from messages from the same client
  and server
id [ CLIENT_HOST DPSS.SERV ]

# messages with the same DPSS.SERV get the same color
group DPSS.SERV

[ +APP_SENT +DPSS_SERV_IN +DPSS_START_READ
+DPSS_END_READ +DPSS_START_WRITE +APP_RECEIVE ]
```

# Network Time Protocol

- For NetLogger timestamps to be meaningful, all systems clocks must be synchronized.
  - NTP is used to synchronize time of all hosts in the system.
    - NTP is from Dave Mills, U. of Delaware (http://www.eecis.udel.edu/~ntp/)
  - Must have NTP running on one or more primary servers, and on a number of local-net hosts, acting as secondary time servers
  - typically get clock synchronized to within 1 millisecond of each other

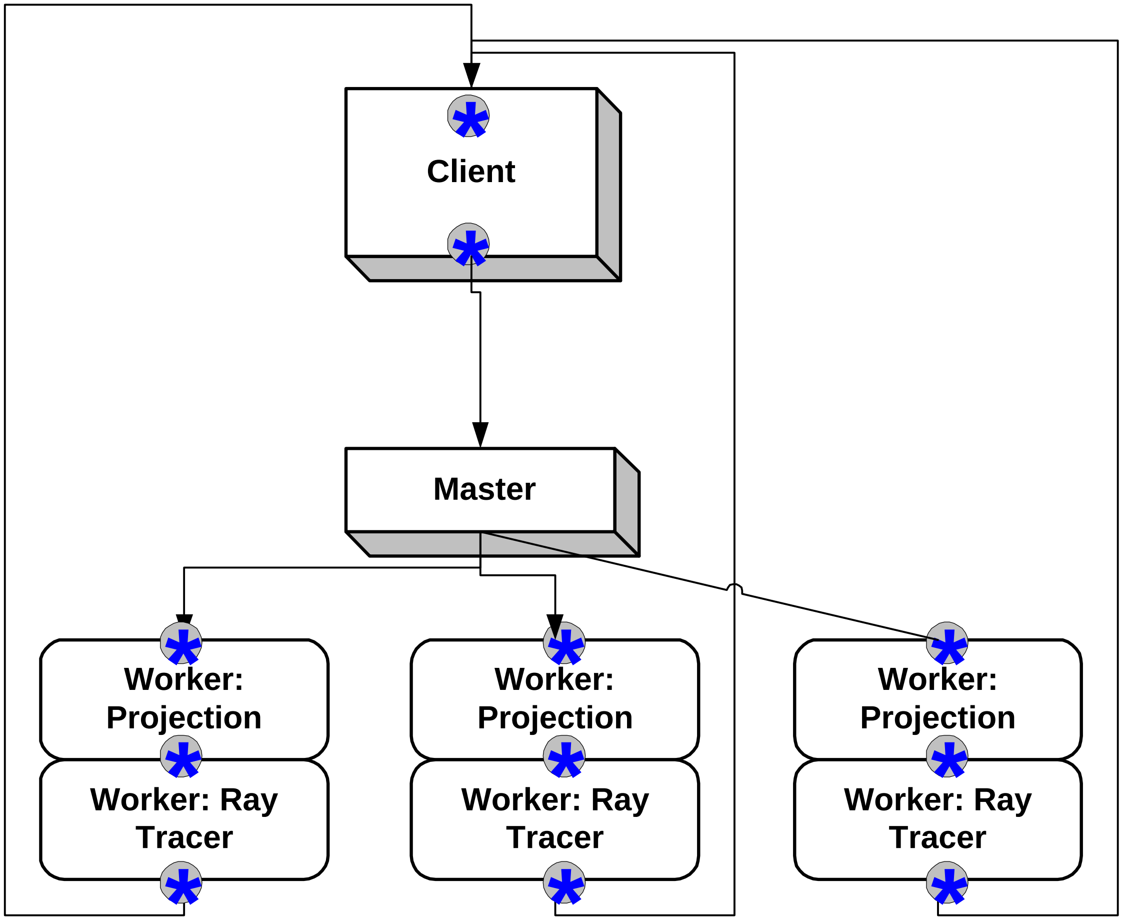
# How to Instrument Your Application

- You’ll probably want to add a NetLogger event to the following places in your distributed application:
  - before and after all disk I/O
  - before and after all network I/O
  - entering and leaving each distributed component
  - before and after any significant computation
    - e.g.: an FFT operation
  - before and after any significant graphics call
    - e.g.: certain CPU intensive OpenGL calls
- This is usually an iterative process
  - add more NetLogger events as you zero in on the bottleneck

# Example 1: Parallel Visualization Application

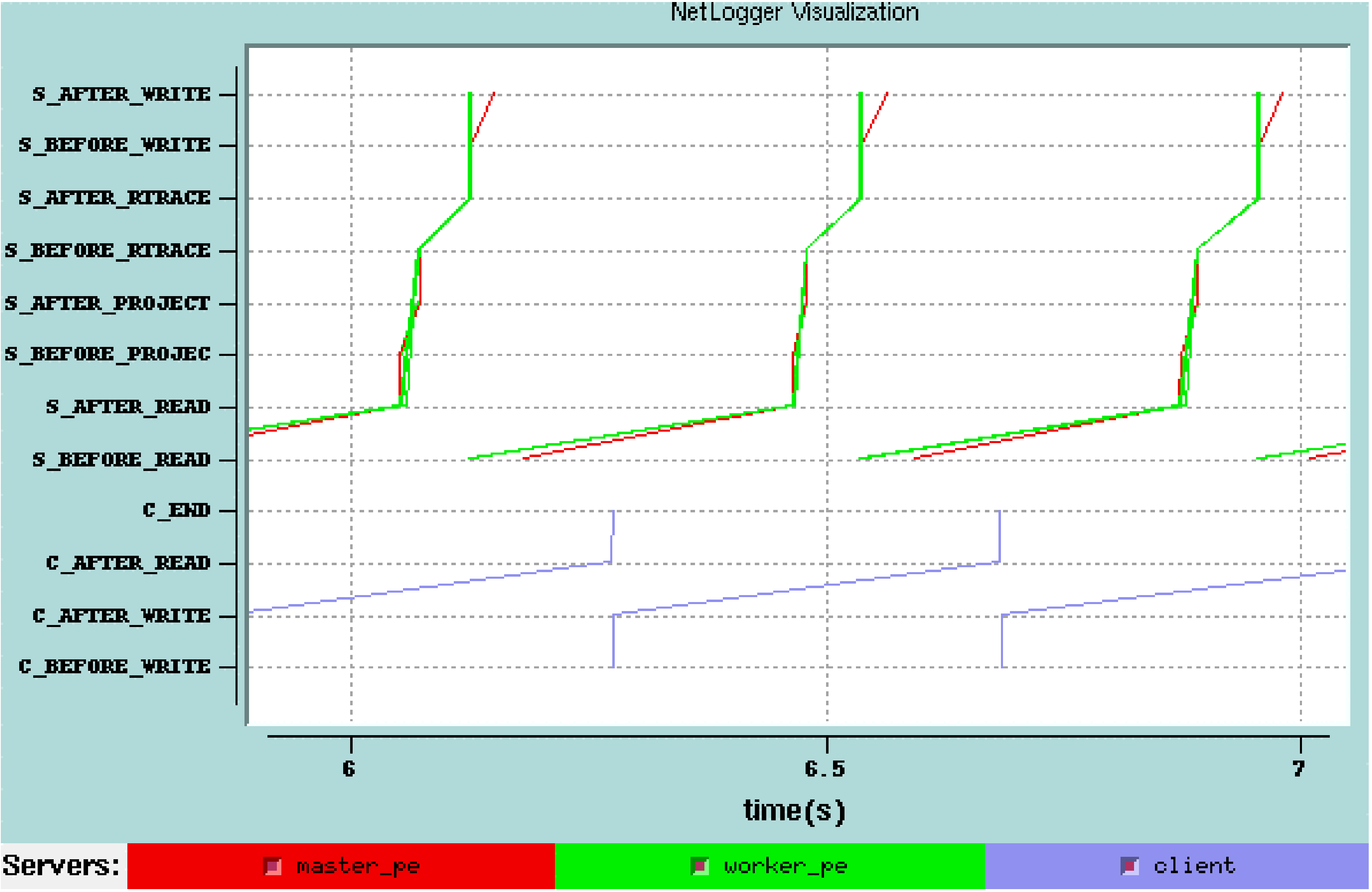
- Radiance is a suite of programs for the analysis and visualization of lighting in design.
  - Input includes the scene geometry, materials, luminance, time, date, and sky conditions
- Radiance has been adapted at LBNL to run on multiple UNIX workstations
  - The image is broken into many small pieces, and illumination calculations are performed for each piece independently
- Used NetLogger to measure:
  - overall system throughput,
  - latency for each stage of getting data, processing it, and writing it
  - patterns of latency which reflect resource contention and other interaction delays

# Radiance Instrumentation Points

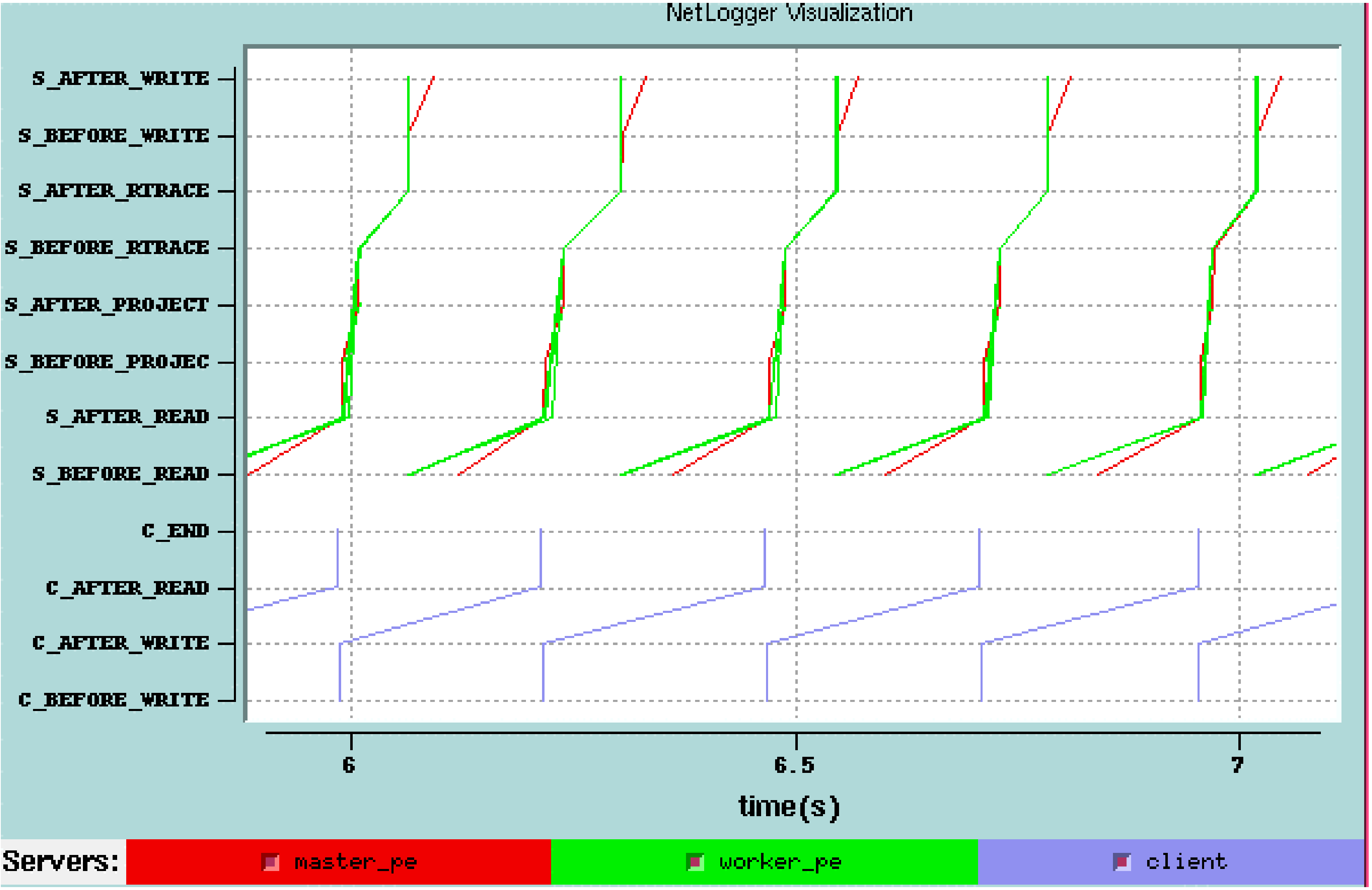


✱ = monitoring point

# NetLogger Radiance Results: Before Tuning



# NetLogger Radiance Results: After Tuning

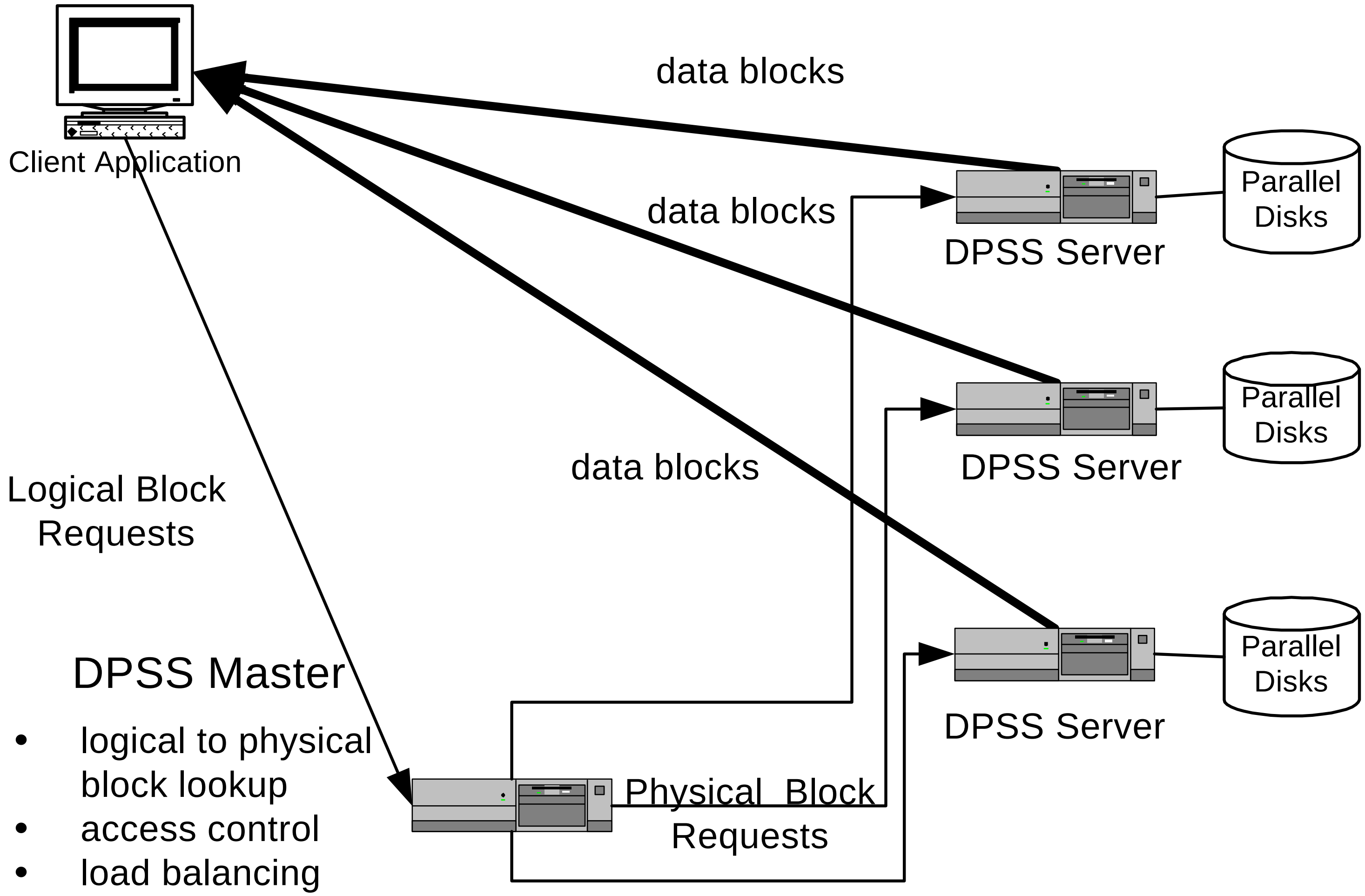


# Example 2: Parallel Data Block Server

- The Distributed Parallel Storage Server (DPSS)
  - provides high-speed parallel access to remote data
  - Unique features of the DPSS:
    - On a high-speed network, can actually access remote data faster that from a local disk
      - 57 MB/sec vs 10 MB/sec

- NetLogger was used for performance tuning and debugging of the DPSS

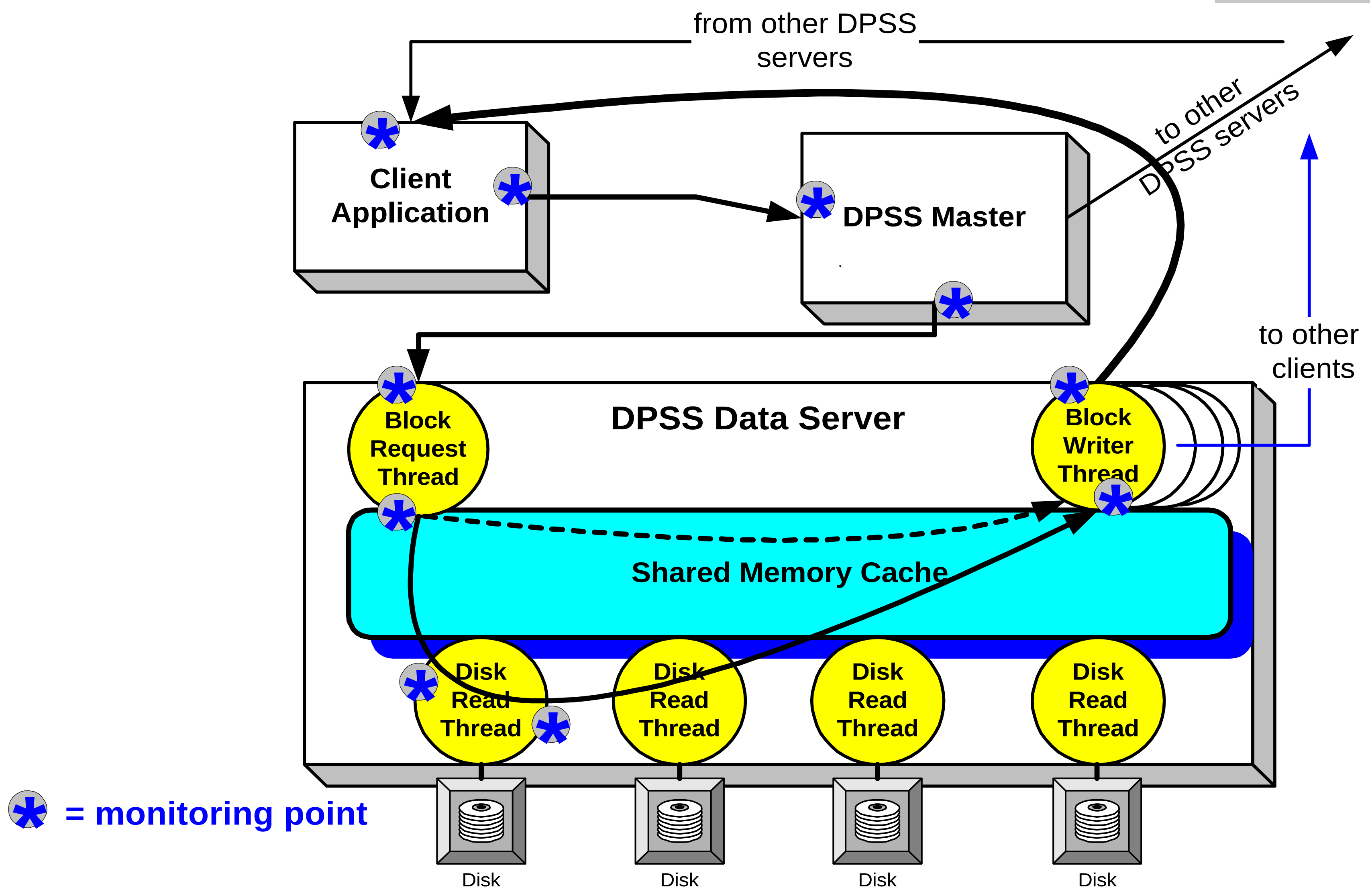
# DPSS Cache Architecture



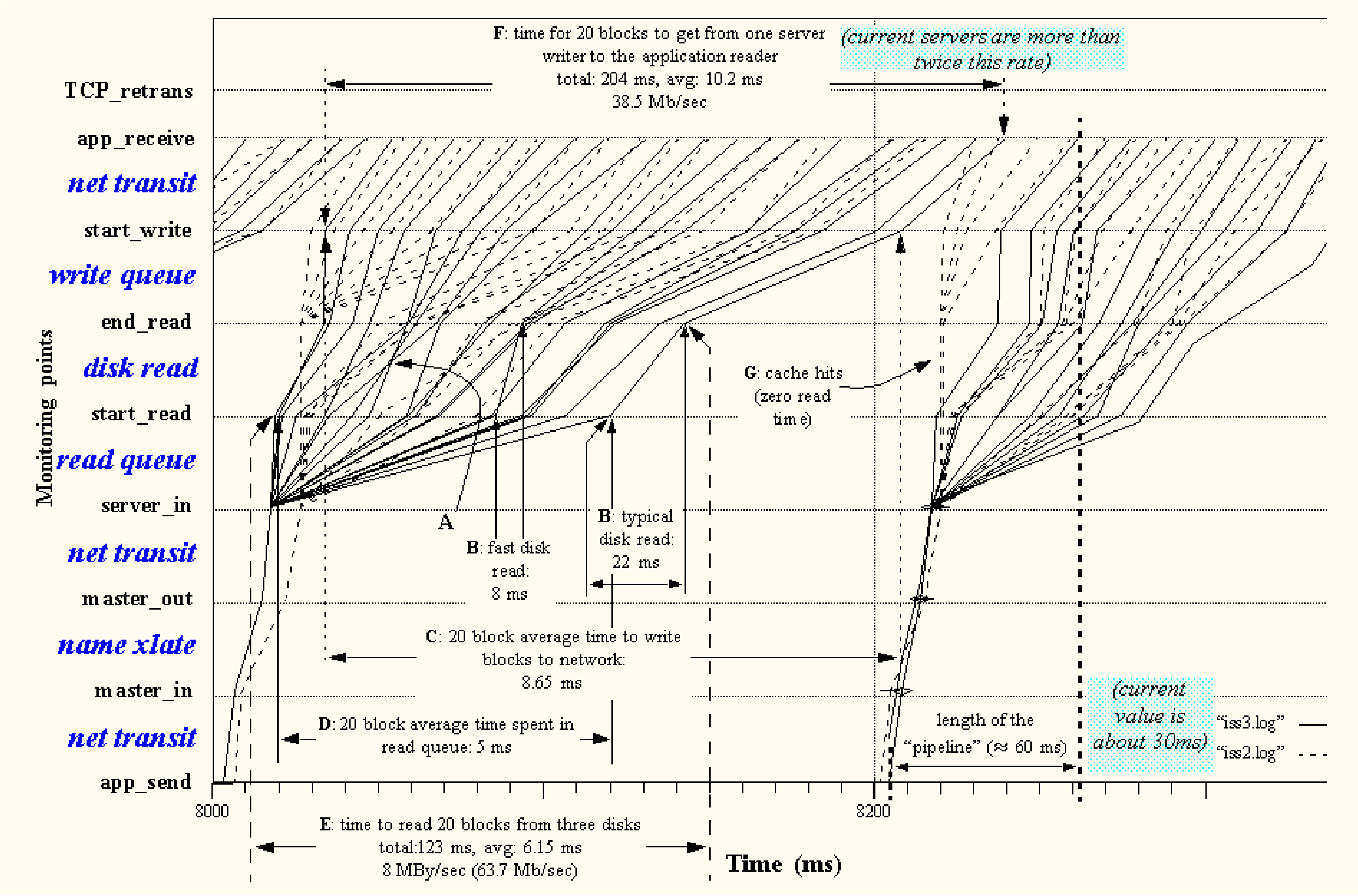
DPSS Master

- logical to physical block lookup
- access control
- load balancing

# DPSS Instrumentation



# NetLogger Results for the DPSS



F: time for 20 blocks to get from one server writer to the application reader total: 204 ms, avg: 10.2 ms 38.5 Mb/sec

*(current servers are more than twice this rate)*

G: cache hits (zero read time)

B: fast disk read: 8 ms

B: typical disk read: 22 ms

C: 20 block average time to write blocks to network: 8.65 ms

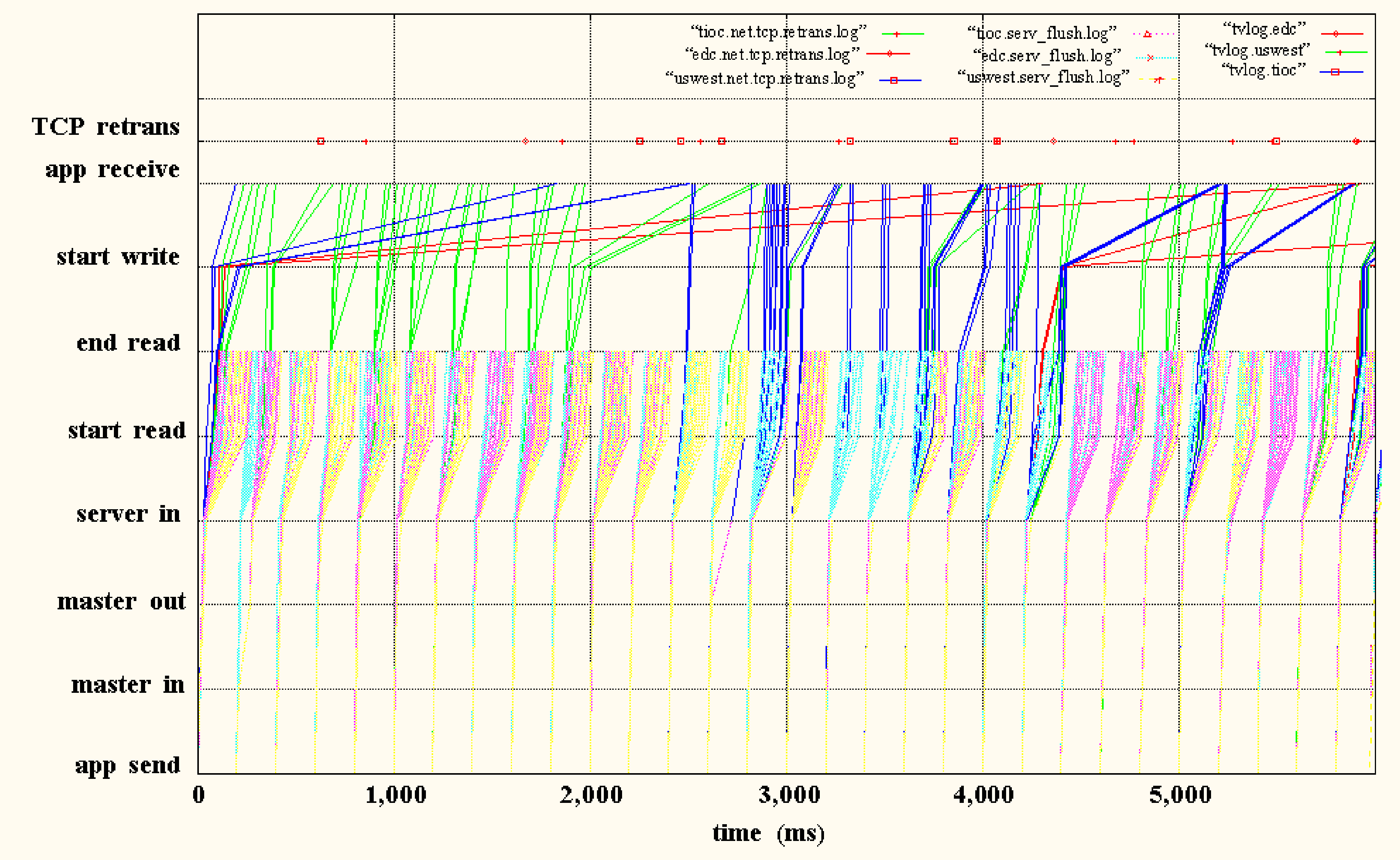
D: 20 block average time spent in read queue: 5 ms

length of the "pipeline" (≈ 60 ms)

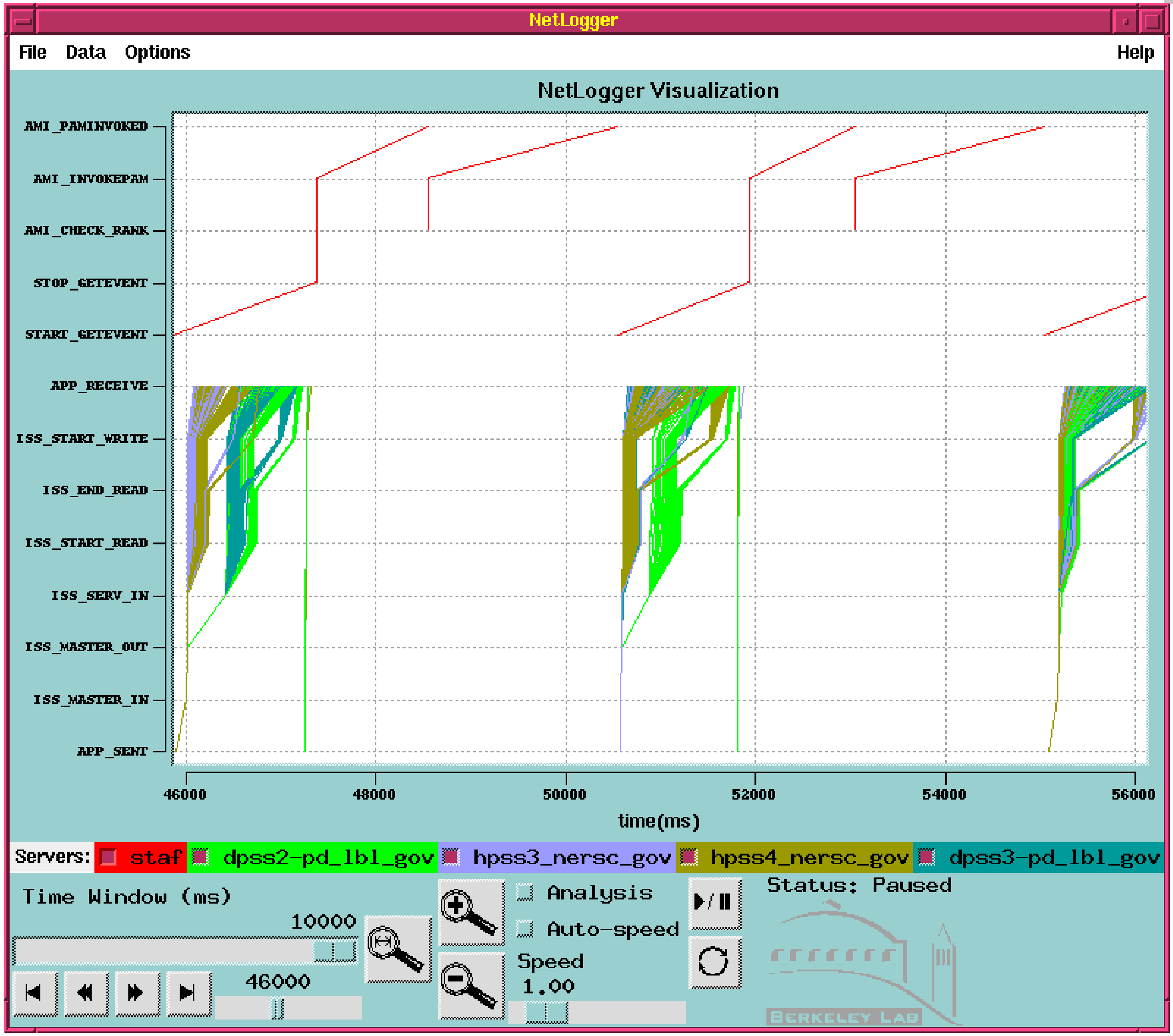
*(current value is about 30ms)*

E: time to read 20 blocks from three disks total:123 ms, avg: 6.15 ms 8 MBy/sec (63.7 Mb/sec)

# NetLogger Results for the DPSS over a WAN



# NLV of DPSS with a HENP client



# Current Work: JAMM

- Java Agents for monitoring and management (JAMM)
  - Java RMI-based agents are used to start up NetLogger versions of system tools
    - netstat, vmstat, uptime, xntpdc, ping, netperf, etc.

- Monitoring can be based on application use
  - e.g.: only do monitoring while a client is connected to a server

- For more info see: http://www-didc.lbl.gov/JAMM/

# Current Work

- NetLogger enhancements:
  - adding Globus security
    - plan to use GlobusIO for sending NetLogger messages to netlogd
  - binary transmission/storage format

- Deployment plan
  - SNMP-based monitoring goes on all the time
  - application/host monitoring triggered by the application/user

# Open Issues

- Log collection/archive service
  - netlogd to a file not adequate, need to send monitoring data to some kind of database (LDAP?)
- multicast ability?
  - Need to simultaneously send to archive and to one or more nlv session
- how to correlate archived monitoring data with network configuration data? (i.e.: traceroute)
- how to map application traffic to a specific switch/router port?
- Integration with other tools
  - Pablo, NWS, Surveyor, etc.

# Getting NetLogger

- Source code and some precompiled binaries are available at:
  - http://www-didc.lbl.gov/NetLogger

- Solaris, Linux, and Irix versions of nlv are currently supported